Microsoft

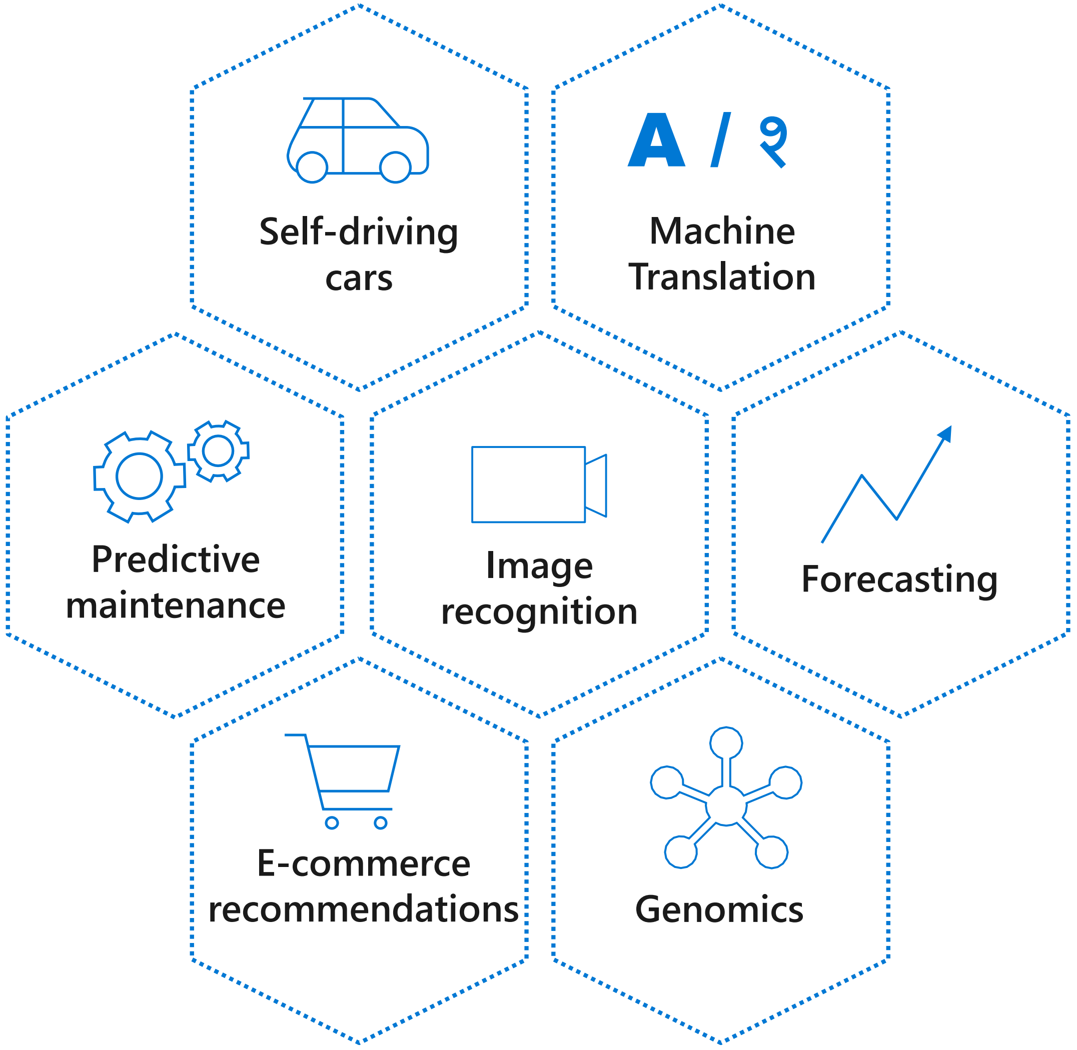
Deep Learning with Microsoft Azure:

Margaryta Ostapchuk
Nnanna Orieke
Sabrina Smai

Table of Contents

Machine Learning

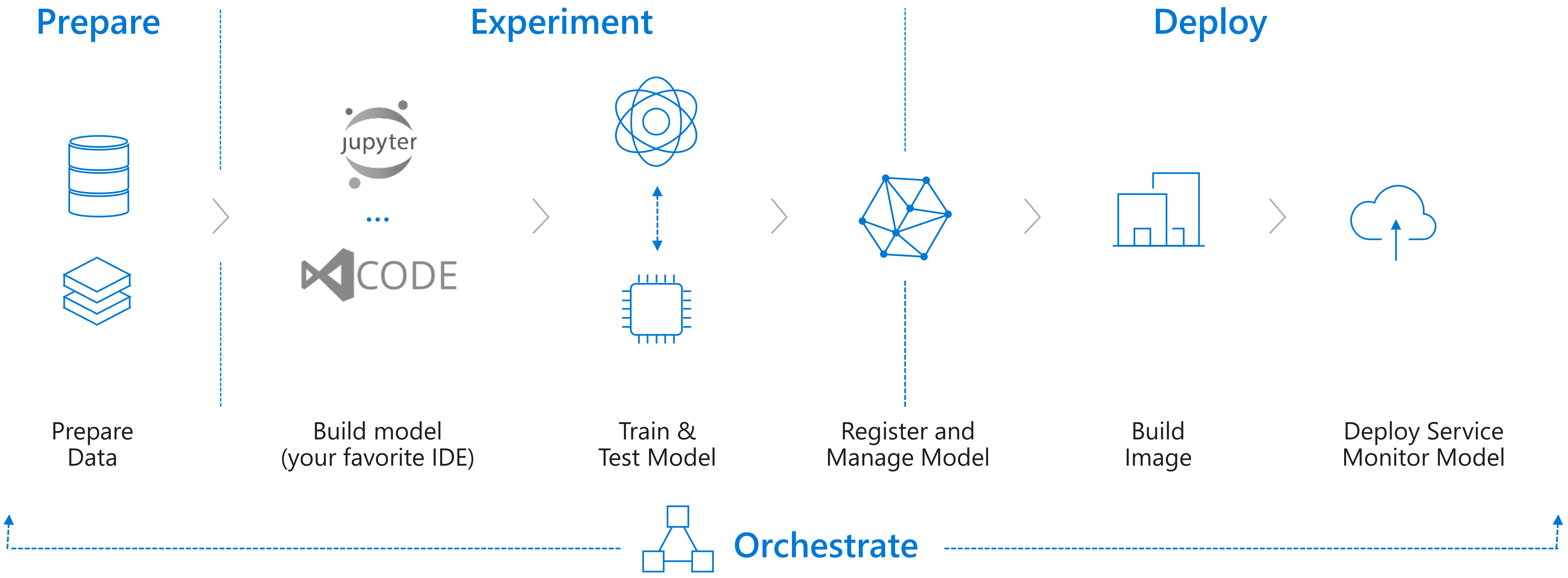
Applications



Requirements of an advanced ML Platform

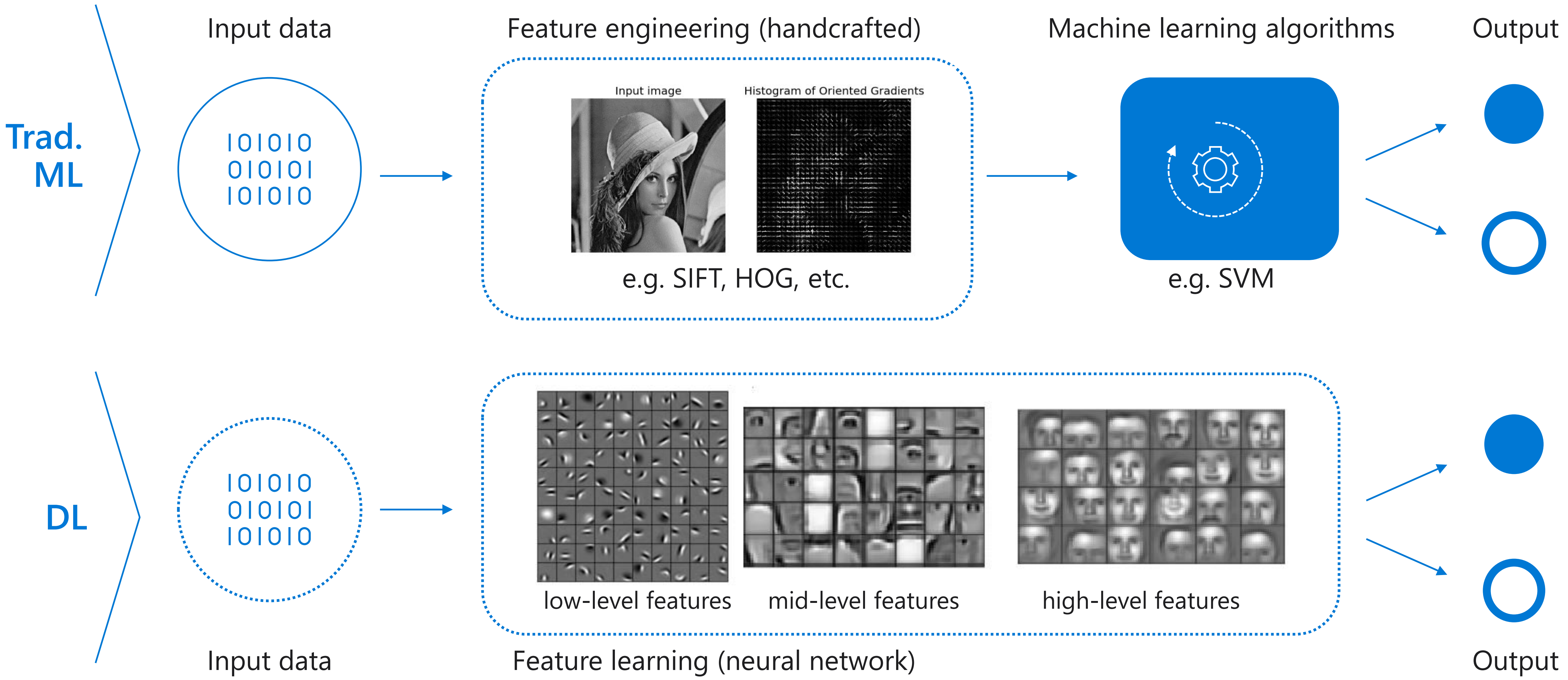
Machine Learning

Typical E2E Process



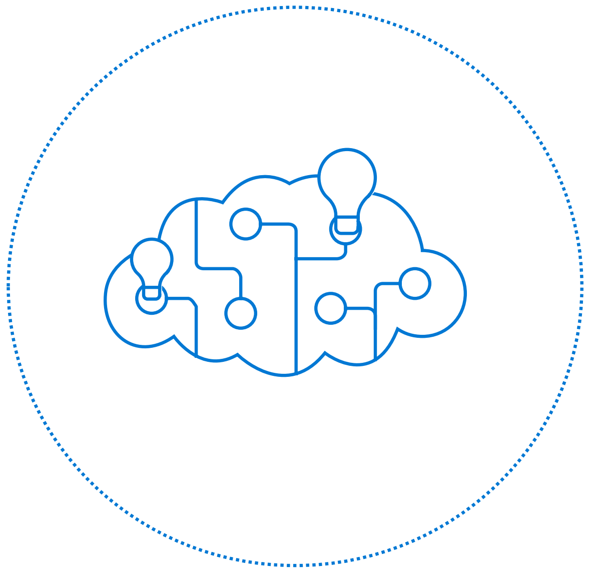
Deep Learning places additional requirements

Traditional ML versus DL



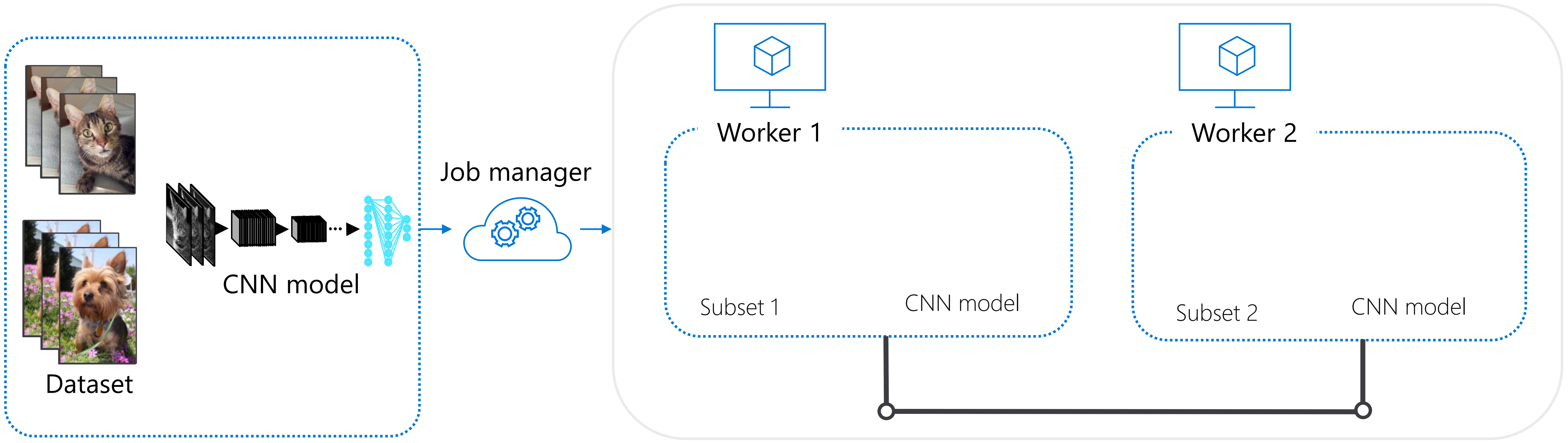
Top figure source;
Bottom figure from NVIDIA

Characteristics of Deep Learning

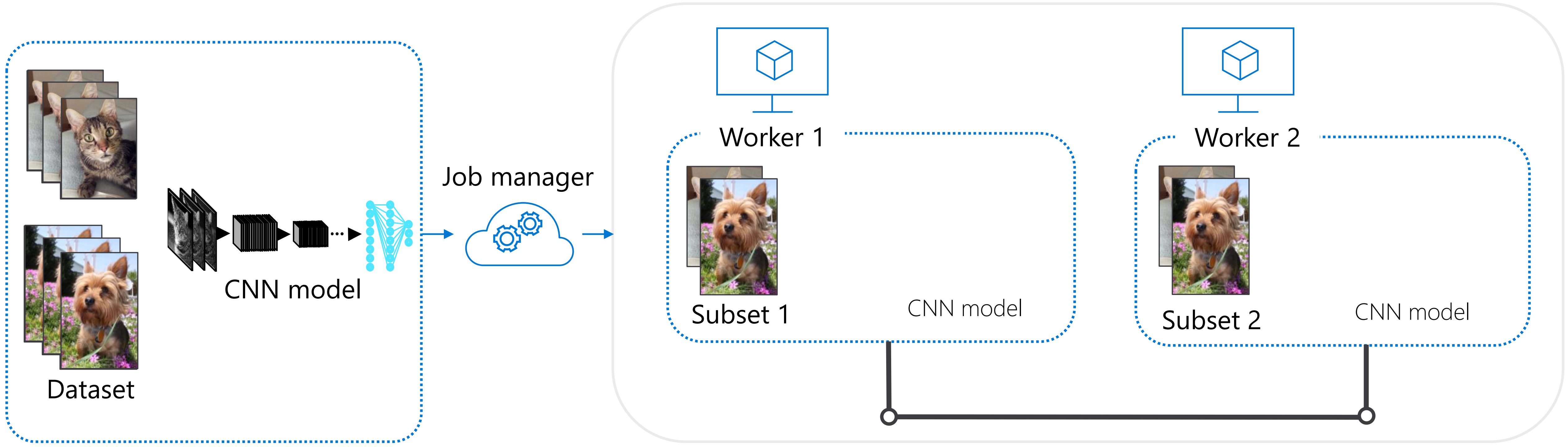


- Massive amounts of training data
- Excels with raw, unstructured data
- Automatic feature extraction
- Computationally expensive

Distributed training mode: Data parallelism



Distributed training mode: Model parallelism



Required Deep Learning Frameworks

Popular Deep Learning Frameworks

TensorFlow

PyTorch

Scikit-Learn

MXNet

Chainer

Keras

ONNX

Community project created by Facebook and Microsoft

Use the best tool for the job. Train in one framework and transfer to another for inference

NVIDIA QUALCOMM Windows vespa SOPHON

intel skymizer MathWorks NETRON Visual DL

Deep Learning

Three Additional Requirements

1. Distributed Training on multi-node clusters
2. Support for advanced processors: TPUs GPUs FPGAs
3. Support for Deep Learning Frameworks:

Azure offers a comprehensive AI/ML platform that meets—and exceeds—requirements

Machine Learning on Azure

Domain specific pretrained models To reduce time to market	Vision	Speech	Language	Search
Familiar Data Science tools To simplify model development	PyCharm	Jupyter	Visual Studio Code	Command line
Popular frameworks To build advanced deep learning solutions	Pytorch	TensorFlow	Scikit-Learn	Onnx
Productive services To empower data science and development teams	Azure Databricks	Azure Machine Learning	Machine Learning VMs	
Powerful infrastructure To accelerate deep learning	CPU	GPU	FPGA	

From the Intelligent Cloud to the Intelligent Edge

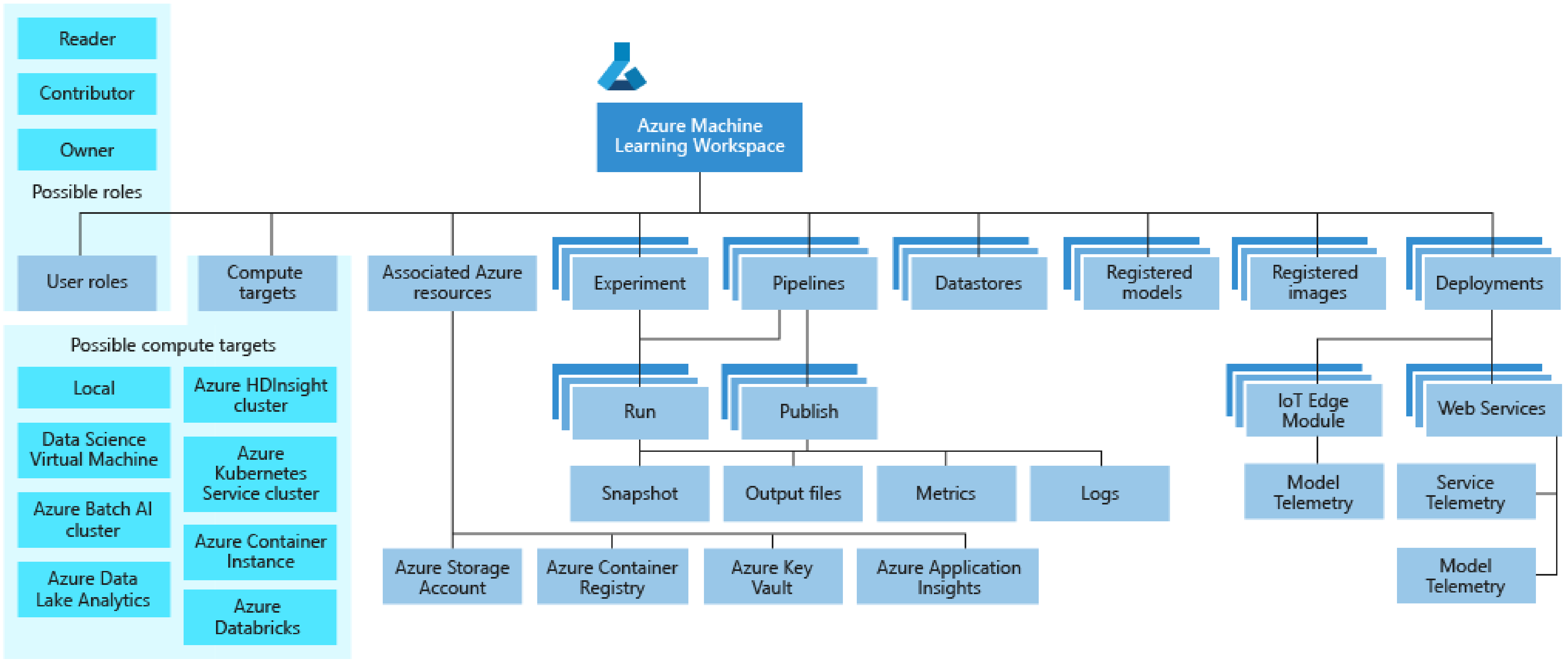
What is Azure Machine Learning service?

Set of Azure Cloud Services + Python SDK + R SDK

That enables you to:

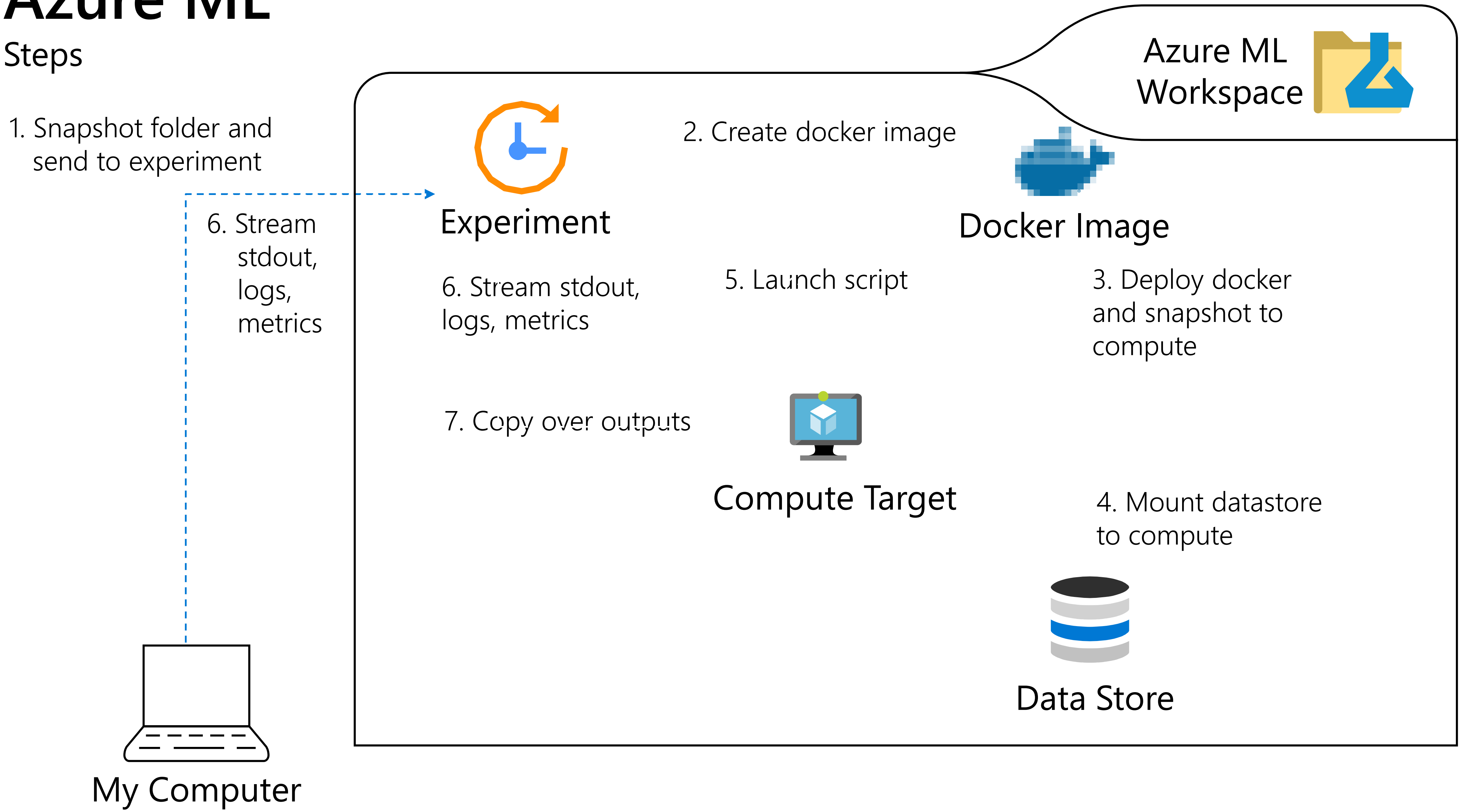
- ✓ Prepare Data
- ✓ Build Models
- ✓ Train Models
- ✓ Manage Models
- ✓ Track Experiments
- ✓ Deploy Models

Azure ML service Workspace Taxonomy



Azure ML

Steps


1. Snapshot folder and send to experiment
6. Stream stdout, logs, metrics
Azure ML Workspace
Experiment
2. Create docker image
Docker Image
6. Stream stdout, logs, metrics
5. Launch script
3. Deploy docker and snapshot to compute
7. Copy over outputs
Compute Target
4. Mount datastore to compute
Data Store
My Computer

Step 1 – Create a workspace

```
from azureml.core import Workspace
ws = Workspace.create(name='myworkspace',
                      subscription_id='<azure-subscription-id>',
                      resource_group='myresourcegroup',
                      create_resource_group=True,
                      location='eastus2' # or other supported Azure region
                     )

# see workspace details
ws.get_details()
```

Step 2 – Create an Experiment

Create an experiment to track the runs in the workspace. A workspace can have multiple experiments

```
experiment_name = ‘my-experiment-1'

from azureml.core import Experiment
exp = Experiment(workspace=ws, name=experiment_name)
```

Step 3 – Create remote compute target

```
# choose a name for your cluster, specify min and max nodes
compute_name = os.environ.get("BATCHAI_CLUSTER_NAME", "cpucluster")
compute_min_nodes = os.environ.get("BATCHAI_CLUSTER_MIN_NODES", 0)
compute_max_nodes = os.environ.get("BATCHAI_CLUSTER_MAX_NODES", 4)

# This example uses CPU VM. For using GPU VM, set SKU to STANDARD_NC6
vm_size = os.environ.get("BATCHAI_CLUSTER_SKU", "STANDARD_D2_V2")

provisioning_config = AmlCompute.provisioning_configuration(
                              vm_size = vm_size,
                              min_nodes = compute_min_nodes,
                              max_nodes = compute_max_nodes)
# create the cluster
print(‘ creating a new compute target... ')
compute_target = ComputeTarget.create(ws, compute_name, provisioning_config)

# You can poll for a minimum number of nodes and for a specific timeout.
# if no min node count is provided it will use the scale settings for the cluster
compute_target.wait_for_completion(show_output=True,
                                   min_node_count=None, timeout_in_minutes=20)
```

Zero is the default. If min is zero then the cluster is automatically deleted when no jobs are running on it.

Step 4 – Upload data to the cloud

First load the compressed files into numpy arrays. Note the '*load_data*' is a custom function that simply parses the compressed files into numpy arrays.

```
# note that while loading, we are shrinking the intensity values (X) from 0-255 to 0-1 so that the
model converge faster.
X_train = load_data('./data/train-images.gz', False) / 255.0
y_train = load_data('./data/train-labels.gz', True).reshape(-1)

X_test = load_data('./data/test-images.gz', False) / 255.0
y_test = load_data('./data/test-labels.gz', True).reshape(-1)
```

Now make the data accessible remotely by uploading that data from your local machine into Azure so it can be accessed for remote training. The files are uploaded into a directory named mnist at the root of the datastore.

```
ds = ws.get_default_datastore()
print(ds.datastore_type, ds.account_name, ds.container_name)

ds.upload(src_dir='./data', target_path='mnist', overwrite=True, show_progress=True)
```

We now have everything you need to start training a model.

Step 5 – Train a local model

Train a simple logistic regression model using scikit-learn locally. This should take a minute or two.

```
%%time from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)

# Next, make predictions using the test set and calculate the accuracy
y_hat = clf.predict(X_test)
print(np.average(y_hat == y_test))
```

You should see the local model accuracy displayed. [It should be a number like 0.915]

Step 6 – Train model on remote cluster

To submit a training job to a remote you have to perform the following tasks:

- 6.1: Create a directory
- 6.2: Create a training script
- 6.3: Create an estimator object
- 6.4: Submit the job

Step 6.1 – Create a directory

Create a directory to deliver the required code from your computer to the remote resource.

```
import os
script_folder = './sklearn-mnist' os.makedirs(script_folder, exist_ok=True)
```

Step 6.2 – Create a Training Script (1/2)

```
%%writefile $script_folder/train.py
# load train and test set into numpy arrays
# Note: we scale the pixel intensity values to 0-1 (by dividing it with 255.0) so the model can
# converge faster.
# ‘data_folder’ variable holds the location of the data files (from datastore)
Reg = 0.8 # regularization rate of the logistic regression model.
X_train = load_data(os.path.join(data_folder, 'train-images.gz'), False) / 255.0
X_test  = load_data(os.path.join(data_folder, 'test-images.gz'), False) / 255.0
y_train = load_data(os.path.join(data_folder, 'train-labels.gz'), True).reshape(-1)
y_test  = load_data(os.path.join(data_folder, 'test-labels.gz'), True).reshape(-1)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape, sep = '\n’)

# get hold of the current run
run = Run.get_context()
#Train a logistic regression model with regularizaion rate of’ ‘reg’
clf = LogisticRegression(C=1.0/reg, random_state=42)
clf.fit(X_train, y_train)
```

Step 6.2 – Create a Training Script (2/2)

```
print('Predict the test set’)
y_hat = clf.predict(X_test)

# calculate accuracy on the prediction
acc = np.average(y_hat == y_test)
print('Accuracy is', acc)


run.log('regularization rate', np.float(args.reg))
run.log('accuracy', np.float(acc)) os.makedirs('outputs', exist_ok=True)


# The training script saves the model into a directory named ‘outputs’. Note files saved in the
# outputs folder are automatically uploaded into experiment record. Anything written in this
# directory is automatically uploaded into the workspace.
joblib.dump(value=clf, filename='outputs/sklearn_mnist_model.pkl')
```

Step 6.3 – Create an Estimator

An estimator object is used to submit the run.

The directory that contains the scripts. All the files in this directory are uploaded into the cluster nodes for execution

```
from azureml.train.estimator import Estimator

script_params = { '--data-folder': ds.as_mount(), '--regularization': 0.8 }

est = Estimator(source_directory=script_folder,
                script_params=script_params,
                compute_target=compute_target,
                entry_script='train.py’,
                conda_packages=['scikit-learn'])
```

Name of estimator

Python Packages needed for training

Training Script Name

Compute target (Batch AI in this case)

Parameters required from the training script

Step 6.4 – Submit the job to the cluster for training

```
run = exp.submit(config=est)
run
```

What happens after you submit the job?

Image creation

A Docker image is created matching the Python environment specified by the estimator. The image is uploaded to the workspace. Image creation and uploading takes about 5 minutes.

This happens once for each Python environment since the container is cached for subsequent runs. During image creation, logs are streamed to the run history. You can monitor the image creation progress using these logs.

Scaling

If the remote cluster requires more nodes to execute the run than currently available, additional nodes are added automatically. Scaling typically takes about 5 minutes.

Running

In this stage, the necessary scripts and files are sent to the compute target, then data stores are mounted/copied, then the entry_script is run. While the job is running, stdout and the ./logs directory are streamed to the run history. You can monitor the run's progress using these logs.

Post-Processing

The ./outputs directory of the run is copied over to the run history in your workspace so you can access these results.

Azure ML Concept

Model Management

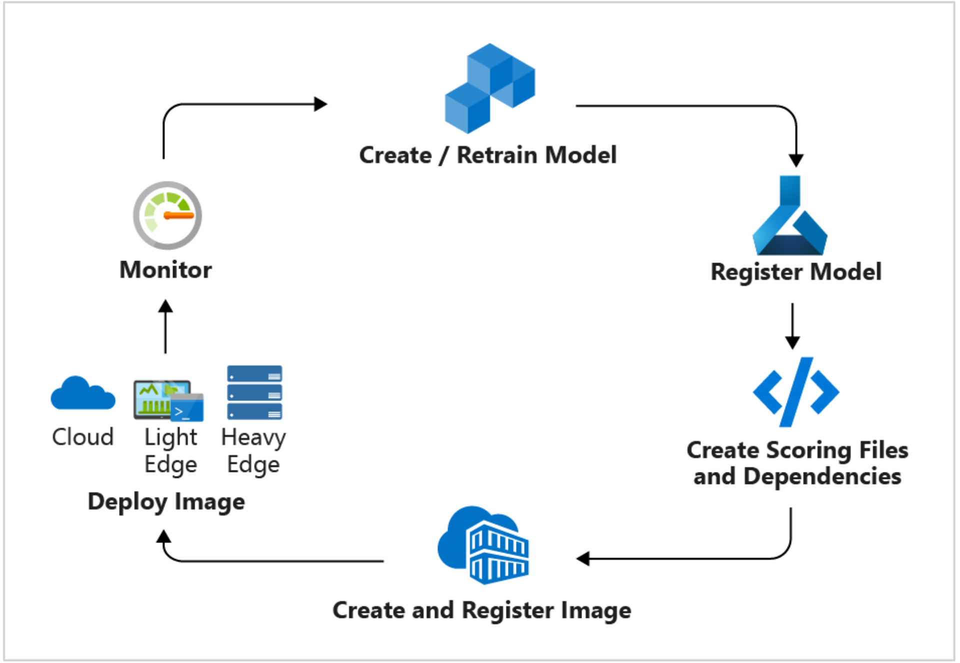
Model Management in Azure ML usually involves these four steps

Step 1: Register Model using the Model Registry

Step 2: Register Image using the Image Registry (the Azure Container Registry)

Step 3: Deploy the Image to cloud or to edge devices

Step 4: Monitor models—you can monitor input, output, and other relevant data from your model.



Azure ML Artifact

Deployment

Deployment is an instantiation of an image. Two options:

Web service

A deployed web service can run on Azure Container Instances, Azure Kubernetes Service, or field-programmable gate arrays (FPGA).

Can receive scoring requests via an exposed a load-balanced, HTTP endpoint.

Can be monitored by collecting Application Insight telemetry and/or model telemetry.

Azure can automatically scale deployments.

IoT Module

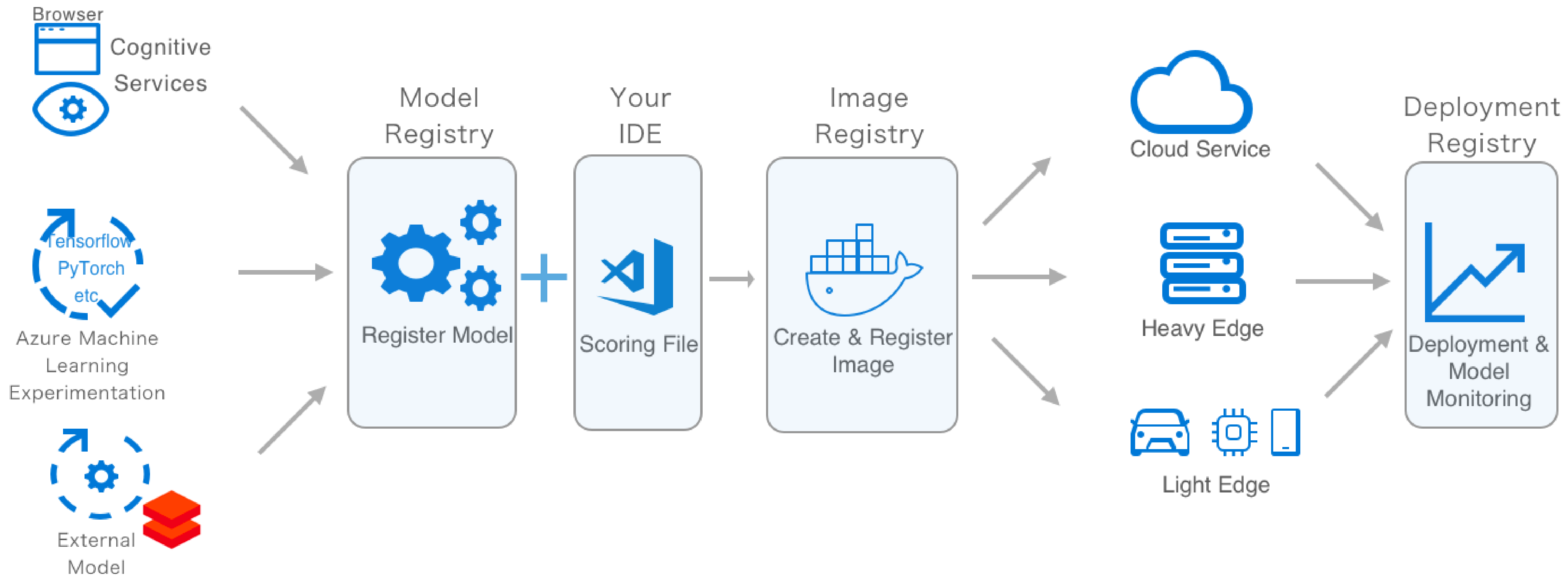
A deployed IoT Module is a Docker container that includes the model, associated script and additional dependencies.

Is deployed using **Azure IoT Edge** on edge devices.

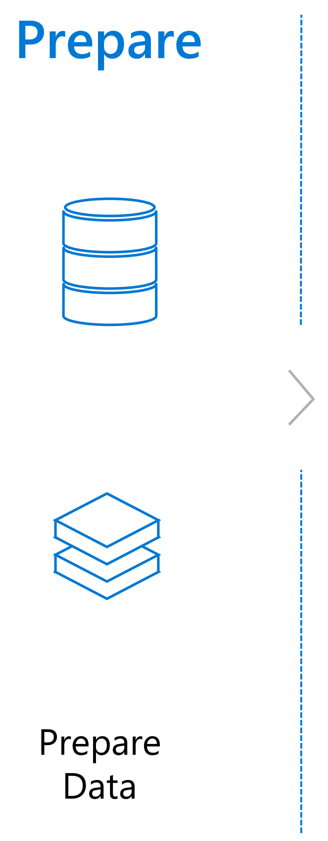
Can be monitored by collecting Application Insight telemetry and/or model telemetry.

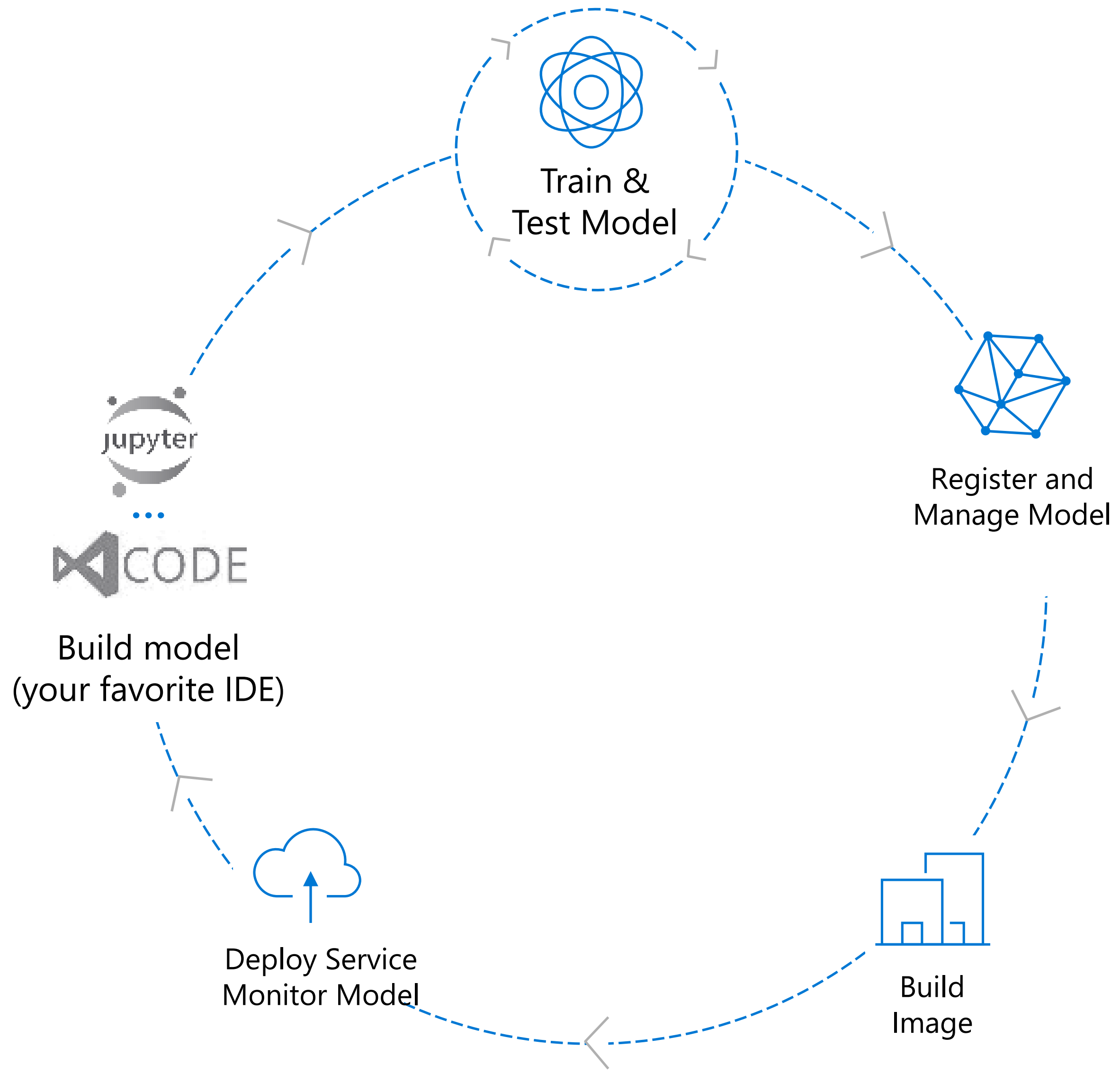
Azure IoT Edge will ensure that your module is running and monitor the device that is hosting it.

Azure ML: How to deploy models at scale



DevOps loop for data science: AML + DevOps




Train & Test Model
Register and Manage Model
Build Image
Deploy Service Monitor Model
jupyter
CODE
Build model (your favorite IDE)

Azure Automated Machine Learning ‘simplifies’ the creation and selection of the optimal model

Automated ML

Current Capabilities

Category		Value
ML Problem Spaces		Classification Regression Forecasting
Frameworks		Scikit Learn
Languages		Python
Data Type and Data Formats		Numerical Text Scikit-learn supported data formats (Numpy, Pandas)
Data sources		Local Files, Azure Blob Storage
Compute Target	Automated Hyperparameter Tuning	Azure ML Compute (Batch AI), Azure Databricks
	Automated Model Selection	Local Compute, Azure ML Compute (Batch AI), Azure Databricks

Distributed Training with Azure ML Compute

Distributed Training with Azure ML Compute

You submit a model training 'job' – the infrastructure is managed for you.

- Jobs run on a native VM or Docker container.
- Supports Low priority (Cheaper) or Dedicated (Reliable) VMS.

Auto-scales: Just specify min and max number of nodes.

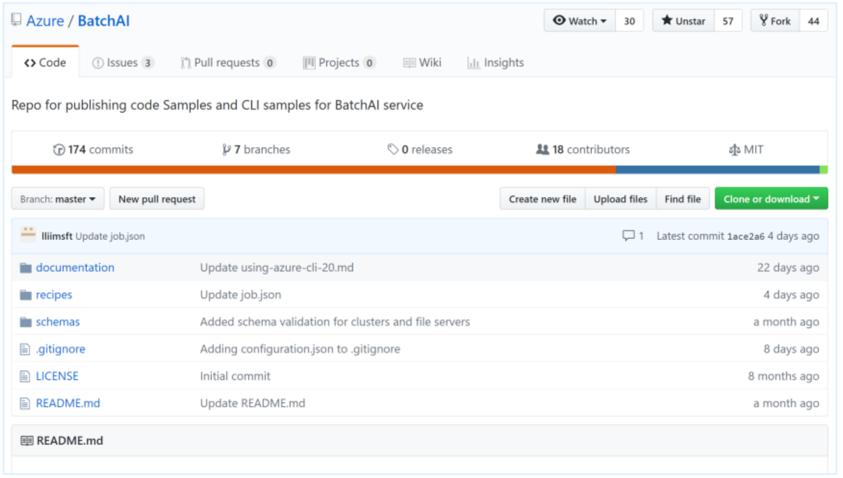
- If min is set to zero, *cluster is deleted when no jobs are running; so pay only for job duration.*

Works with most popular frameworks and multiple languages.

Supports distributed training with Horovod.

Cluster can be shared; multiple experiments can be run in parallel.

Supports most VM Families, including latest NVidia GPUs for DL model training.



Repo for publishing code Samples and CLI samples for BatchAI service

174 commits | 7 branches | 0 releases | 18 contributors | MIT

llimsft Update job.json — 1 — Latest commit 1ace2a6 4 days ago

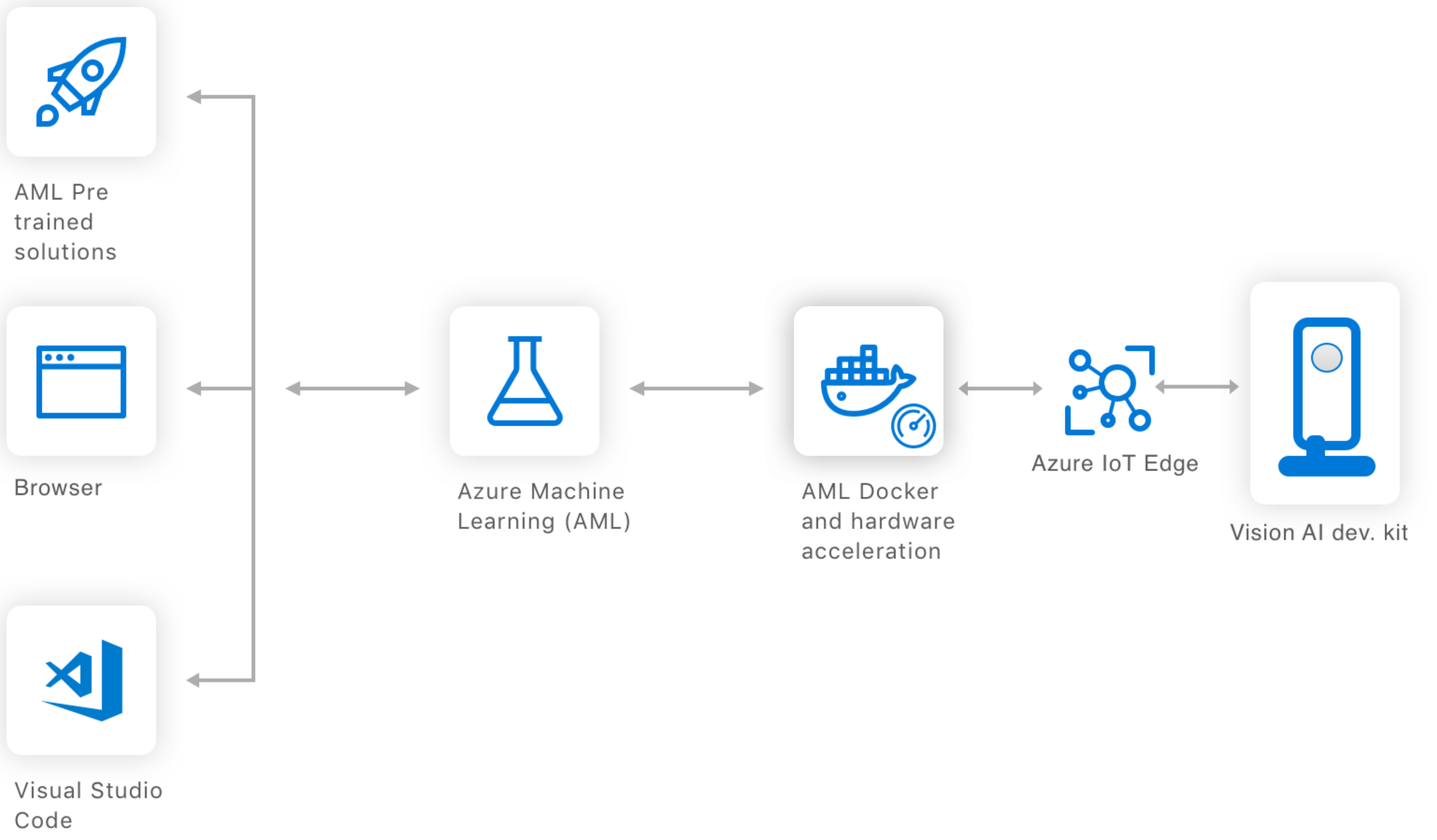
documentation	Update using-azure-cli-20.md	22 days ago
recipes	Update job.json	4 days ago
schemas	Added schema validation for clusters and file servers	a month ago
.gitignore	Adding configuration.json to .gitignore	8 days ago
LICENSE	Initial commit	8 months ago
README.md	Update README.md	a month ago

README.md

Deploy to IoT Edge

Vision AI Development kit

System Architecture


AML Pre trained solutions
Browser
Visual Studio Code
Azure Machine Learning (AML)
AML Docker and hardware acceleration
Azure IoT Edge
Vision AI dev. kit

What Azure Machine Learning Service

- End-to-End ML Life-cycle Management
- Model Interpretability-- IntepretML
- Fairness -- FairLearn
- DataDrift
- MLOPS – Integration with Azure Devops
- Build Anywhere, Manage and Deploy with AML
- Access to Compute on Demand
- Manage cost with Pipeline Reuse
- Low Priority VM
- Fiull Supports OSS and Enterprise

Demo